データベースアウトソーシングにおける問合せ解析からの社会的情報漏洩防止

AUTHOR(S):

川本, 淳平; 吉川, 正俊

CITATION:

川本, 淳平 ...[et al]. データベースアウトソーシングにおける問合せ解析からの社会的情報漏洩防止. 情報処理学会研究報告. データベースシステム(DBS) 2008, 2008(88): 79-84: IPSJ-DBS08146014.

ISSUE DATE:

2008-09-14

URL:

http://hdl.handle.net/2433/147389

データベースアウトソーシングにおける問合せ解析からの社会的情報漏洩防止

川本　淳平†　　吉川　正俊†

† 京都大学大学院情報学研究科　〒 606–8501 京都市左京区吉田本町
E-mail: †j.kawamoto@db.soc.i.kyoto-u.ac.jp, ††yoshikawa@i.kyoto-u.ac.jp

あらまし　近年，管理すべきデータ量の増加に伴いデータベースのアウトソーシングサービスが普及しつつある．データがサービスプロバイダ側で管理されるこのサービスでは，今まで以上にセキュリティが重要な問題となっている．サービスプロバイダによるデータ閲覧を防止するために，サーバへ保存するデータをクライアント上で予め暗号化しておくことは，こうしたセキュリティ問題を解決する一つの手段であり多くの研究が行われている．しかし，データの暗号化だけでは十分なセキュリティが確保できるとは言い難い．問合せやその結果を解析することで，ユーザ間の関係やコミュニティにおける重要度などの社会的情報をサービスプロバイダは得ることができる．本論文では，こうした問合せ解析によるユーザの社会的情報の漏洩を防ぐために，問合せやその結果に対する匿名性を確保するための技術について論じる．

キーワード　データベースセキュリティ，プライバシー保護，社会的情報

Securing Social Information from Query Analysis in Outsourced Databases

Junpei KAWAMOTO† and Masatoshi YOSHIKAWA†

† Graduate School of Informatics, Kyoto University
Yoshida–Honmachi, Sakyo–ku, Kyoto, 606–8501 Japan
E-mail: †j.kawamoto@db.soc.i.kyoto-u.ac.jp, ††yoshikawa@i.kyoto-u.ac.jp

Abstract　Nowadays, outsourced databases services are in widespread use. In these services, security is more important than ever to, because user data are stored and managed by service providers. To prevent the service providers from inspection of user data, many techniques are introduced such as encryption data at the client and how to execute queries over encrypted data etc. However, only data encryption does not provide enough security. Query and the result analysis enable service providers to find social information such as relations between users, community and roles of them. In this paper, we introduce anonymity for the query and the result to secure our social information.

Key words　Database Security, Privacy Preservation, Social Information

1. はじめに

近年，企業や組織においてデータベースシステムで管理しなければならないデータ量は急速に増加しており，これらの膨大なデータを安全に管理することは社会にとって重要な課題となっている．他方，いつでもどこにいても膨大な情報を利用することができるユビキタスコンピューティング環境が整備されつつあり，従来の閉じたネットワークからだけでなく，外出先のネットワークやモバイル端末からウェブを通じてデータベースシステムにアクセスしたいという要求も高まっている．このことはつまり，データベースに対して，今まで以上に高いセキュリティと可用性が求められていることを意味している．なぜなら，閉じたネットワークとは異なり，オープンな環境からのアクセスを許可した場合，悪意あるユーザからの攻撃など様々な脅威に対処せねばならないからである．

こうした状況の中で，データベースとその管理を外部へ移譲するデータベースのホスティングサービスが普及してきており，Amazon S3 [1] や Salesforce [2] などのサービスが提供されている．組織のネットワークとは切り離されたサービスプロバイダ上にデータベースを置くことで，ネットワークのセキュリティレベルを低下させず，またプロバイダにサーバ管理を移譲するため，低いコストで高いセキュリティと可用性を持つデー

タベースを利用することができる．

しかし，データを外部へ移譲するデータベースのホスティングサービスにより，新たに発生する情報漏洩のリスクも存在する．サービスプロバイダが提供するデータベースサーバは，機密情報保護に関しては常に信用できるとは限らないためである．例えば，サービスプロバイダはサーバに対しあらゆる権限を持ち得るため，外部へ移譲されたデータベースを利用するユーザはサービスプロバイダによるアクセスを制限することは難しい．サーバへ送信するデータを予めクライアント上で暗号化しておく暗号化データベースはサーバに対し望まないアクセスを禁じる有効な手段である [3]．この暗号化データベースに関しては，大きく分けて次の 2 種類の研究がなされている．

（1） 暗号化されたデータにセキュアインデックスと呼ばれるオリジナルデータの断片を付加することにより，暗号化されたままのデータに対して問合せの高速処理を行う方法 [4], [5], [6].

（2） 複数の鍵を使い分けてデータを暗号化することにより，従来のデータベースシステムが提供する機能とは別のレイヤでアクセスコントロールを実現する方法 [7].

これらの研究は，機密情報の保護手段としては有効であるが，それだけでは不十分である．暗号化によりデータの内容に関する安全性は保証できるが，データに付加されるアクセス権限情報や，問合せ及びその結果の解析により様々な情報が漏洩する危険性がある．特に，ユーザがどのデータを誰と共有しているのかを解析することで，ユーザどうしの知り合い関係や繋がり関係，ユーザ間にどのようなグループが存在しその中心ユーザは誰なのか，また組織におけるユーザの階層構造などといった，各ユーザの社会的情報を調べることができる．

本論文では，こうしたユーザの社会的情報漏洩を防ぐために，問合せ解析による社会的情報漏洩を防ぐ問合せ一般化と取得データ一般化を提案する．問合せからの社会的情報漏洩とは，同じ問合せを行うユーザが限られている場合，そのユーザは何らかの関係がある可能性が高いというものである．この情報の解析を防ぐために，動的ハッシュを用いていくつかの異なる問合せをまとめ，サーバ側には同一の問合せと見せかける．これにより，特定の問合せとユーザの対応付けを防ぐ．また，返却データからの漏洩とは，ユーザが取得するデータにはそのユーザはアクセス権限を持っているというものである．従って，いくらアクセス権限情報を暗号化していても，ユーザに返されるデータを解析することで権限情報が判明する可能性がある．そのため，ブルームフィルタを用いて，各データを少なくとも k 人に取得させる．このようにすることで，アクセス権限の推定に k 匿名性を保証することができる．

本論文の構成は，続く 2 節にて問合せ及び取得データの一般化を行うシナリオについて説明する．その後，3 節にて問合せ一般化の方法について，4 節で取得データ一般化の方法について説明し，最後に 5 節でまとめと今後の課題について述べる．

2. 基本シナリオ

本節では，社会的情報漏洩を防ぎアウトソーシングデータベースを利用するための基本的なシナリオについて説明する．

表 1 暗号前のオリジナルテーブル: *schedule*

No.	name	begin	end	acl
1	Products Review	7/17 15:00	7/17 18:00	Alice, Bob
2	Business Trip	7/18 10:00	7/20 18:00	Alice, Bob
3	Team Meeting	7/20 15:00	7/20 18:00	Alice, Carol
4	Business Trip	7/21 12:00	7/21 17:00	Dave

表 2 サーバに保存される暗号化テーブル： $schedule^*$

No.	*etuple*	I_{name}	I_{begin}	I_{end}	BF
1	5f0f1f46...	00	10	10	$B_{\{Alice,Bob\}}$
2	b98009af...	01	00	11	$B_{\{Alice,Bob\}}$
3	082ba604...	10	11	11	$B_{\{Alice,Carol\}}$
4	8bc546af...	01	01	01	$B_{\{Dave\}}$

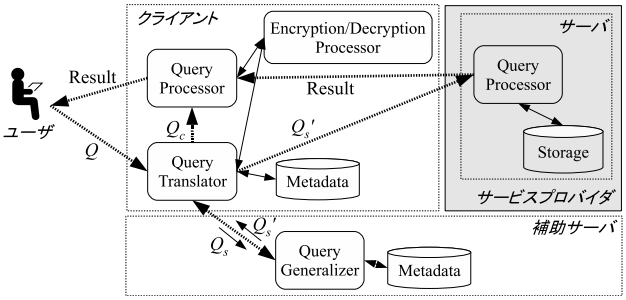
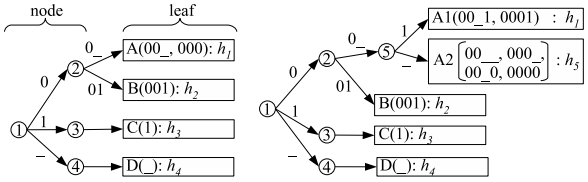


図 1 問合せと取得データ一般化を行うシナリオ

本論文では，表 1 に示すテーブルをアウトソーシングデータベースに保存することを考える．データ自身の漏洩を防ぐために，このテーブルはに示す暗号化テーブルへと変換されてサーバへ保存されているものとする．表における，属性 *etuple* は元のタプルを暗号化したものである．また，I_{name}，I_{begin} 及び I_{end} は，暗号化されたタプルに対する問合せ処理に用いられる元データの断片でありセキュアインデックスと呼ばれる．このセキュアインデックスの作成と問合せでの利用法は様々な種類が提案されており，Wang らのハッシュを用いたもの [4]，Hacıgümüş らによるバケットを用いたもの [5]，Damiani らによる B+木を用いたもの [6] が代表的である．本論文では $I_{name}, I_{begin}, I_{end}$ の全てにハッシュインデックスを用いる．また，ハッシュ値のビット文字列を使って表すものとする．したがって，例えば，“00” $= hash($“Products Review”$)$ となる．他のインデックス形式を用いてもかまわない．属性 BF は，ブルームフィルタを用いたアクセスコントロールリストのセキュアインデックスであり，詳細は 4. 節にて説明する．

次に，図 1 を用いてテーブル $schedule^*$ に対する問合せシナリオについて説明する．図において，クライアントは各ユーザ毎に用意され，アウトソーシングデータベースに接続する．また，補助サーバは，少ない記憶容量と計算能力しか持たないが信頼できるサーバである．この時，ユーザによる，オリジナルスキーマ上での問合せ Q は次のように実行される．

（1） クライアントは Q を Query Translator に渡しサーバ用の問合せ Q_s 及び，サーバからの検索結果を復号したデータへの事後処理用問合せ Q_c を生成する．また，



(a) 動的ハッシュ　　(b) 葉の分割

図 2　動的ハッシュの概要

Q_s を補助サーバ上の Query Generalizer へ渡す．

（2） Query Generalizer は問合せを一般化する．

（3） クライアントは補助サーバから一般化された問合せ Q'_s を受け取りサーバへ問い合わせる．

（4） サーバは，受け取った Q'_s を実行し，暗号化されたままの結果をクライアントに返却する．

（5） クライアントはサーバから受け取ったデータを復号し，復号化データに対して Q_c を実行し不要なタプルを取り除く．最後に最終結果をユーザに返却する．

3. 動的ハッシュによる問合せ一般化

前述のように，サーバは特定の問合せとユーザの対応付けを解析することで，ユーザ間のグループ情報を推測することができる．本節では，問合せとユーザの対応付けを防ぐ問合せ一般化について説明する．問合せ一般化では，いくつかの異なる問合せをまとめ，サーバに対しては同じ問合せであるように見せかける，問合せを一般化することで，必ず異なるグループがサーバに対して同じ問合せを行うようになり，問合せ解析からサーバがグループ情報を特定することを防ぐことができる．

本研究では，この問合せ一般化に動的ハッシュ [8] を用いる．動的ハッシュとは，データを木構造を用いて管理するデータ構造であり，各葉に保存されるデータの分布が一定になるように格納される．この動的ハッシュを用いて，ユーザが行った問合せを記録していくことで，問合せと，その問合せを行ったユーザ集合の分布が一定に保たれるように一般化することができる．

3.1 動的ハッシュの概要

本論文で用いる動的ハッシュは，図 2(a) に示すようにノードと葉からなるデータ構造である．ノードは他のノードあるいは葉へのラベル付きポインタを保持しており，葉は複数のハッシュ値を格納している．以下では，ハッシュ値を {0, 1, _ } からなる文字列として扱う．

各ハッシュ値は，ポインタのラベルを元に格納先が決まる．例えば，表 3 のハッシュ文字列 h_1〜h_4 を格納すると，図 2(a) のようになる．h_1 = “00_11_” の格納手順について見てみると，まず，ルートノードから出ている各ポインタのラベルと h_1 の先頭文字列 “0” との比較を行う．その結果，ノード 2 へ進む．次に，ノード 2 から出ているポインタと，h_1 のうち先ほど比較に使用した文字列を除いた文字列を比較し葉 A へ到着する．

なお，ラベル中の “_” はワイルドカードであり，ハッシュ文字列の {0, 1, _ } 全てにマッチする．複数のラベルにマッチする場合，“_” を含まないラベルを優先するものとする．図 2(a) 及び図 2(b) において，葉に記されている文字列集合は，この文字列から始まるハッシュが格納されることを表している．すなわち，例えば図 2(a) の葉 A には，“00_” または，“000” で始まるハッシュが格納される．

表 3　条件節の例とそのハッシュ文字列

No.	条件節	ハッシュ文字列
h_1	$I_{name} = 01$ & $I_{begin} = 01$	00_11_
h_2	$I_{name} = 01$ & $I_{begin} = 00$ & $I_{end} = 11$	001101
h_3	$I_{name} = 10$ & $I_{end} = 11$	1_10_1
h_4	$I_{begin} = 10$ & $I_{end} = 10$	_11_00
h_5	$I_{name} = 00$ & $I_{begin} = 00$	00_00_

動的ハッシュでは，葉に格納されるハッシュ値の個数が一定数を超えるなどの条件を満足した場合，葉の分割が行われる．分割は必要に応じて新たにノードを追加し，ポインタを張り直すことで行う．ただし，ポインタを張り直す場合，ポインタラベルのいずれかには必ず “_”を含むものとする．

例えば，葉に含まれるハッシュ値が 2 個になった場合に分割を行う場合を考え，図 2(a) に示す動的ハッシュに表 3 の h_5 を追加するとする．この時，上記の格納先決定ルールに従うと h_5 は葉 A に格納される．よって，葉 A に含まれるハッシュ値が 2 個になり葉の分割が行われる．この分割では新たにノード 5 が追加され，分割された葉 A1，A2 に格納されるハッシュ値が 1 個ずつになるようにポインタを張り直される．この条件を満たすラベルの設定方法には，(“1”, “0”), (“1”, “_”), (“0”, “_”) の 3 通り考えられる．しかし，いずれかのラベルには “_”を含まなければならないため，ここでは，(“1”, “_”) を選択している．したがって，分割後の動的ハッシュは，図 2(b) のようになる．

3.2 動的ハッシュ利用のための準備

本節では，問合せ一般化に動的ハッシュを用いるために必要ないくつかの値について説明する．

問合せ先のテーブルに a 個の属性 $A_1, A_2, \cdots, A_a$ があるとする．ある問合せ q に含まれる，属性 A_i のセキュアインデックス I_i の値を $I_i(q)$ と書き，これをビット列と見た場合の j ビット目を $[I_i(q)]_j$ と書くことにする．簡単のためビット列は全て長さ n とするが，短いビット列があれば 0 を追加すれば良い．また，I_i が問合せ q に現れない場合，$[I_i(q)]_j$ = “_” $(1 \leqq j \leqq n)$ とする．これらを用いて，問合せ q に対し動的ハッシュの作成に用いるハッシュ:$H(q)$ を図 3 に示す文字列として定義する．つまり，問合せ中の各セキュアインデックスをビット列文字列として扱い，全属性の 1 ビット目を順に並べ，その後 2 ビット目を並べるというようにビットごとに並べ直したものである．

例えば，表 2. に対する問合せ，

$$q_s : \text{SELECT } * \quad \text{FROM } \ schedule^* \quad \text{WHERE } I_{name} = \text{“01”} \text{ AND } I_{begin} = \text{“01”} \qquad (1)$$

に対して $H(q_s)$ を求める．この問合せには，I_{end} が現れていないので，$[I_{end}(q_s)]_j$ = “_” $(j = 1, 2)$ と見なす．したがって，

$$H(q) = \boxed{[I_1(q)]_1 \mid [I_2(q)]_1 \mid \cdots\cdots \mid [I_a(q)]_1 \mid [I_1(q)]_2 \mid [I_2(q)]_2 \mid \cdots\cdots \mid [I_n(q)]_a}$$

図 3　問合せ一般化に用いる問合せ q のハッシュ文字列 $H(q)$.

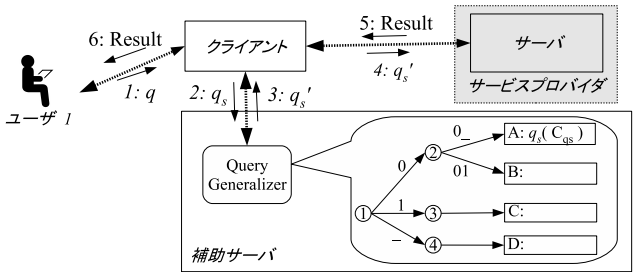
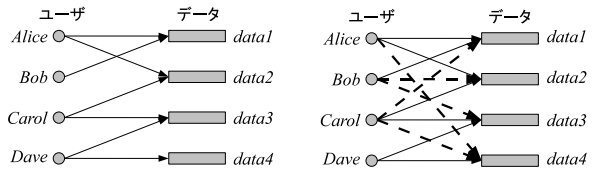


表 4　問合せ一般化の流れ



(a) 通常の取得関係　　(b) 一般化後の取得関係

図 4　データ取得情報からの権限情報推定

$H(q_s)$ = “00_11_” となる．表 3 はいくつかの問合せ条件節に対しハッシュH を求めたものである．

また，総ユーザ数を N としユーザ集合の集合を格納するためのカウンタとして，ハッシュ値 $H(q)$ ごとに N 要素ビット列 C を用意する．

$$C_{H(q)} = \begin{array}{|c|c|c|c|c|}\hline 0 & 1 & 0 & \cdots\cdots & 1 \\ \hline \end{array}$$
$$u_1 \quad u_2 \quad u_3 \qquad\qquad u_N$$

このカウンタへのユーザの追加は，ユーザに対応するビットに 1 をセットすることで行う．例えば，u_1 を追加する場合，配列の 1 番目の要素の値に 1 をセットする．

3.3　問合せ一般化

ユーザ u が問合せ Q を行い，クライアントにより問合せ Q_s が Query Generalizer に渡されたとする．このとき，一般化は次のように行われる．

（1） 問合せ Q_s からハッシュ$H(Q_s)$ を計算する．
（2） 動的ハッシュを辿り $H(Q_s)$ が該当する葉を求める．葉に $H(Q_s)$ が格納されていなければ追加する．
（3） カウンタ $C(H(Q_s))$ にユーザ u を追加する．
（4） ルートノードから $H(Q_s)$ が格納される葉に至るパス上のラベルを連結した文字列 l を得る．
（5） 得られた文字列 l を図 3 により各属性に対するセキュアインデックスへ分割する．この時，文字列 l の長さが足りない場合は末尾に “_”を付加する．

図 4 は，動的ハッシュが図 2(a) である場合に，ユーザ 1 の

q : SELECT ∗
　FROM　*schedule*
WHERE　*name* = “Business Trip”
　　AND *begin* = “7/21 12:00”

という問合せを一般化する手順について表している．クライアントは，問合せ q を基に式 (1) で表わされる q_s を作成し補助サーバ上の Query Generalizer に送る．Query Generalizer は，この q_s のハッシュを作成し動的ハッシュを探索する．前述のように $H(q_s)$ = “00_11_” であるから，この問合せ q_s は葉 A に該当することになる．よって，ルートから葉 A に至るパス上のラベルを連結し，“00_” を得る．$H(q_0)$ の文字数は 6 文字であり不足しているため “_”を補い，l = “00____” となる．得られた l を図 3 を用いて条件節に分割すると次のようになる．

I_{name} LIKE “0_”AND I_{begin} LIKE “0_”AND I_{end} LIKE “__”

SQL 文における “_”は任意の 1 文字とマッチする．したがって，I_{name} LIKE “0_” という条件節は，I_{name} が “00” または “01” の選択を行う．また，I_{end} LIKE “__” という条件節は，全てのインデックスにマッチするため不要であり削除する．よって，以下の一般化された問合せ q_s' をクライアントへ返す．

q_s' : SELECT ∗
　FROM　$schedule^*$
WHERE　I_{name} LIKE “0_” AND I_{begin} LIKE “0_”

クライアントは，この q_s' を用いてサーバに問い合わせる．

また，葉の分割は，各葉に少なくとも $2c$ 個以上のハッシュが格納されており，要素が c 個以上のハッシュ集合 $\mathbf{H_1}$ と $\mathbf{H_2}$ に以下の条件を満足するように分割できる場合に行われる．ここで，c は定数である．任意の 2 ハッシュ$h_1, h_2 \in \mathbf{H_1}$ について，$C(h_1) \cap C(h_2) = \emptyset$ である．また同様に任意の 2 ハッシュ $h_3, h_4 \in \mathbf{H_2}$ についても $C(h_3) \cap C(h_4) = \emptyset$ である．この時，$\mathbf{H_1}$ と $\mathbf{H_2}$ が別々の葉になるように分割する．これはつまり，各葉には全く異なるユーザによる問合せが少なくとも c 個以上格納されていることを表し，また上記の一般化手順により，それらの異なる問合せが同じ問合せとして行われることを表している．したがって，サーバにとっては c 個のグループより詳しい情報を取得することはできない．

4.　ブルームフィルタによる取得データ一般化

各データに設定されているアクセス権限を解析することで，ユーザの社会的情報を推定することが可能である．したがって，このアクセス権限情報は秘匿すべきであるが，問合せの結果としてユーザに返却されるデータを解析することで，間接的に各ユーザがどのデータにアクセス権限を持っているのかを取得することができる．

例えば，問合せとその結果として返却されるデータの関係を

図 4(a) の二部グラフとして考える．図中の矢印は問合せの結果データが返却されることを表す．つまり，Alice による何らかの問合せの結果として $data1$ 及び $data2$ が返却されるということを表わしている．サーバはユーザの問合せに応じてデータを返却しているため，こうした情報を得ることは容易である．このグラフを解析することで，Alice と Bob が $data1$ にアクセス権限を持っていることがわかる．すなわち，$data1$ の権限情報を推測することが可能である．

そこで，サーバによる権限情報の推測に対して匿名性が確保できるように，無関係なデータも含めて取得することを行う．図 4(b) は冗長なデータも混ぜて取得することを表したもので，図中の点線矢印は，無関係なデータをあえて取得していることを表している．この図では，各データとも必ず 3 人が取得している．そのため，サーバから見ると実際にアクセス権限を持っているユーザは各々3 人まで絞れるが特定することはできない．本研究では，この取得データからの権限情報推測に対して，ブルームフィルタ [9] を用いた権限情報のセキュアインデックスを作成することで k 匿名性を保証する．以降では，先ずブルームフィルタを用いたセキュアインデックスについて説明したのち，k 匿名性を保証するインデックスの作成方法について述べる．

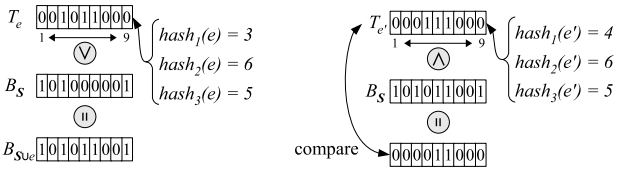
4.1 ブルームフィルタによるセキュアインデックス

ブルームフィルタは，集合の記憶に用いられるデータ構造の一種である．ある要素が集合に含まれるか否かの判定に対して，いくらかの偽陽性はあるものの高速に判定することができる．また，記憶容量の面からも効率的なデータ構造である．ブルームフィルタ $BF_{n,k}$ は，集合を表す n ビットのビット列 B と，要素の追加や判定に使用する k 個の異なるハッシュ関数からなる．ハッシュ関数の値域は 1 から n の整数である．また，空集合を表す場合ビット列 $B_\emptyset$ は各ビットに 0 がセットされる．

集合 $\mathbf{S}$ を表すビット列 $B_\mathbf{S}$ へ要素 e を追加する場合は次のようにする．先ず，k 個のハッシュ関数を用いて追加要素 e から k 個の整数を計算する．次に，得られた整数の位置に 1 をセットした長さ n のビット列 T_e を用意する．したがって，T_e には k 個の 1 がセットされることになる．新しいビット列 $B_{\mathbf{S}\cup e}$ は，$B_\mathbf{S}$ において T_e で 1 がセットされている位置に 1 をセットしたものである．これを形式的に表すと，$B_\mathbf{S}$ と T_e のビットごとの論理和を取ったものであり，$B_{\mathbf{S}\cup\mathbf{e}} = B_\mathbf{S} \vee T_e$ となる．

また，ある値 e が集合 $\mathbf{S}$ の元であるか否かの判定は次のように行う．先ず，要素の追加と同様にして調べる要素に対してハッシュ関数を用いて T_e を作成する．要素の追加方法から，要素 e が $\mathbf{S}$ に含まれていれば，T_e において 1 がセットされている箇所は全て，$B_\mathbf{S}$ においても 1 である．逆に，一つでも 0 である場合，$\mathbf{S}$ は e を含まないと判定できる．したがって，要素の判定を形式的に表すと，条件式 $T_e = B_\mathbf{S} \wedge T_e$ が成り立たない場合，要素 e は集合 $\mathbf{S}$ に含まれていないといえる．

図 5(a) は，ブルームフィルタ $BF_{9,3}$ のある集合 $\mathbf{S}$ に要素 e を追加する場合を表している．図では，三つのハッシュ関数を用いて追加する要素 e から三つの整数 $\{3,5,6\}$ を得ている．したがって T_e は，これら三つの番号に 1 がセットされた長さ 9 のビット列であり，$B_\mathbf{S}$ に追加すると $B_{\mathbf{S}\cup e}$ のようになる．



(a) 要素の追加　　(b) 要素の判定

図 5　ブルームフィルタを用いたセキュアインデックス

図 5(b) は，ブルームフィルタ $BF_{9,3}$ のある集合 $\mathbf{S}$ に対し要素 e' を含んでいるかを判定している．先ず，要素の追加と同様に三つのハッシュ関数を用いて $T_{e'}$ を作成している．次に，対象の集合 $\mathbf{S}$ を表すブルームフィルタ $B_\mathbf{S}$ とビットごとの論理積を求める．図では，得られたビット列は $T_{e'}$ と異なっており，$T_{e'} = B_\mathbf{S} \wedge T_{e'}$ は成り立たない．したがって e' は集合 $\mathbf{S}$ に含まれていないと判定できる．

このブルームフィルタを用いた権限情報のセキュアインデックスの作成方法は，極めてシンプルである．アクセス許可されたユーザ集合を表すビット列を作成しインデックスとして用いれば良い．例えば，Alice と Bob がアクセス権限を持つデータがあるとすると，Alice と Bob のユーザ ID をそれぞれハッシュ関数に渡して T_{Alice}, T_{Bob} を計算する．これらを基に，上記要素の追加手順を行い $B_{\{Alice,Bob\}} = B_\emptyset \vee T_{Alice} \vee T_{Bob}$ を求める．こうして得られた $B_{\{Alice,Bob\}}$ をインデックスとして付加する．表 2. の属性 BF は，このブルームフィルタインデックスを表わしている．また，インデックスを用いた問合せは次にように行う．問い合わせるユーザを u とすると，ユーザ u のクライアントはハッシュ関数を用いて T_u を作成する．次に，サーバ用の問合せ Q_s の条件節に $T_u = BF \wedge T_u$ を追加する．つまり，アクセス権限を持つユーザ集合にユーザ u が含まれているタプルのみを取得している．このようにして，サーバに対し権限情報を秘匿したままサーバ上での権限判定を行う．

4.2 匿名性を保証する改良型ブルームフィルタインデックス

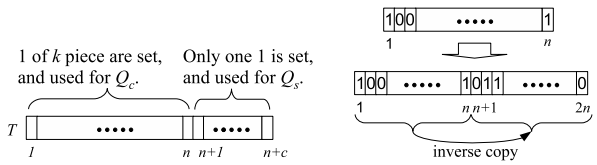
前節で説明したブルームフィルタインデックスを用いれば，権限情報を明示的に付加せずに，サーバ上での権限判定を行うことが可能になる．しかし，前述の通りユーザへ返却されるデータを解析することで間接的に権限情報を推測することは可能である．本節では，ブルームフィルタインデックスを改良し，間接的な権限情報推測を防止する．

改良型ブルームフィルタ $BF'_{n,k,c,\phi}$ は，通常のブルームフィルタ $BF_{n,k}$ より c ビット長い，$n+c$ ビットのビット列 B と k 個のハッシュ関数を持つ．また，ϕ は $[1, 2(n+c)]$ の置換関数である．このフィルタにおいて，空集合を表すビット列 $B_\emptyset$ は通常のブルームフィルタ同様に全てのビットに 0 がセットされる．

この改良型ブルームフィルタでは，要素の追加に値域が $[1, n]$ の k 個の異なるハッシュ関数と以下の関数 f を共に用いる．

$$f(x) = (x \bmod c) + c + 1 \qquad (2)$$

ここで，x はユーザの ID であり，f は $[n+1, n+c]$ の値を返



(a) 問合せに用いる T　　(b) 01 個数の平滑化

図 6　改良型ブルームフィルタを用いたセキュアインデックス

す．したがって，要素の追加に用いられるビット列 T には，図 6(a) に示すように，1 ビット目から n ビット目までに k 個の 1 がセットされ，$n+1$ ビット目から $n+c$ ビット目までにただ一つの 1 がセットされる．

この改良型ブルームフィルタを先ほど説明した権限情報セキュアインデックスに用いる．インデックスの作成方法は通常のブルームフィルタと同様に，アクセス権限を与えるユーザ集合 $\mathbf{S}$ に対するビット列 $B_{\mathbf{S}}$ を作成すれば良い．インデックスを用いた問合せがこれまでと異なる．問い合わせるユーザを u とすると，ユーザ u のクライアントは関数 f を用いて 1 つの整数のみを得る．そして，得られた整数の位置にのみ 1 をセットしたビット列 T'_u を用意する．サーバ用の問合せ Q_s で用いるのはこの T'_u であり，Q_s の条件節に $T'_u = BF \wedge T'_u$ を追加する．次に，k 個のハッシュ関数を用いて通常のブルームフィルタ同様に T_u を作成する．この T_u は，クライアント用の問合せ Q_c に用い，Q_c の条件節に $T_u = BF \wedge T_u$ を追加する．

改良型ブルームフィルタにおけるハッシュ関数の性質から，$[n+1, n+c]$ の各ビットには，それぞれ $\lceil N/c \rceil$ 人のユーザが割り当てられることになる．$K = \lceil N/c \rceil$ とすると，サーバ上での権限判定の結果，各ユーザは $K-1$ 人分の余分なデータまで取得することになる．その代り，各データともアクセス権限を持つユーザ 1 人に対し，最低 K 人が取得することになる．つまり，権限情報の推測に関して K 匿名性を保証することができる．また，クライアントにおいて本来のブルームフィルタを用いた権限判定を行うことで，実際には権限を持たない余分なデータを取り除いている．

さらに，改良型ブルームフィルタを用いたセキュアインデックス中に含まれる 1 の個数から，アクセス権限を持つユーザ数が推測される可能性があることを考慮し，ビット列中の 0 及び 1 の個数を平滑化し，アクセス権限を持つ人数を推定できないようにする．例えば，アクセス権限を持つユーザがただ一人の場合，セキュアインデックスにおける 1 がセットされたビットの個数は $k+1$ 個となるため，権限を持つユーザ数を特定することができる．ただし，この場合でも，そのデータは最低 K 人が取得することにはなる．0, 1 個数の平滑化は，図 6(b) に示すようにダミービットを追加することで行う．ここでは簡単のため，本来の n ビットに対し 01 を反転したビット列を追加し長さ $2n$ のビット列を作成する．このようにすることで，どのようなビット列であっても 1 がセットされているビットの個数は n 個となる．最後に，このままでは，ダミービットが含まれていることが明らかであるためビット列をシャッフルする．並び替えには，置換関数 ϕ を用いて行う．

以上より，最終的な改良型ブルームフィルタ $BF'_{n,k,c,\phi}$ を用いたセキュアインデックスでは，長さ $2n$ のビット列が各データに付加されてサーバに保存されることになる．そのため，クライアントも問合せを行う場合，ユーザのユーザ ID から n ビット列 T を作成した後，0 を追加して長さを $2n$ ビットへと拡張し，その後置換関数を用いてシャッフルしたビット列を用いて問合せを行う．

5. まとめと今後の課題

本論文では，アウトソーシングデータベースにおいてユーザの社会的情報漏洩を防ぐために次の二つについて提案を行った．一つ目は，問合せ解析から特定の問合せとユーザとの対応関係を秘匿する，動的ハッシュを用いた問合せ一般化である．動的ハッシュを用いて，いくつかの異なる問合せを一つの問合せとして見せかけることで，問合せ情報からユーザ間の関係情報が推定されることを防いでいる．二つ目は，取得データから各データに設定されているアクセス権限情報の解析を防ぐ，ブルームフィルタを用いた取得データ一般化である．実際にはアクセス権限を持っていないデータをいくつか取得することで，取得データからそのデータにアクセス権限を持っているユーザが特定されることを防いでいる．

今後の課題としては，本論文で提案した手法について実際のサービス上で動作するアプリケーションを作成し定性的・定量的な実験を行いその効果を検証することである．

文　献

[1] Amazon S3: http://aws.amazon.com/s3.
[2] Salesforce: http://www.salesforce.com/.
[3] G. I. Davida, D. L. Wells and J. B. Kam: “A database encryption system with subkeys”, ACM Trans. Database Syst., **6**, 2, pp. 312–328 (1981).
[4] Z.-F. Wang, W. Wang and B.-L. Shi: “Fast query over encrypted character data in database”, CIS ’04: Proceedings of the International symposium on computational and information science, Vol. 3314, pp. 1027–1033 (2004).
[5] H. Hacıgümüş, B. Iyer, C. Li and S. Mehrotra: “Executing sql over encrypted data in the database-service-provider model”, SIGMOD ’02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data, New York, NY, USA, ACM, pp. 216–227 (2002).
[6] E. Damiani, S. D. C. Vimercati, S. Jajodia, S. Paraboschi and P. Samarati: “Balancing confidentiality and efficiency in untrusted relational dbmss”, CCS ’03: Proceedings of the 10th ACM conference on Computer and communications security, New York, NY, USA, ACM, pp. 93–102 (2003).
[7] E. Damiani, S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi and P. Samarati: “Key management for multi-user encrypted databases”, StorageSS ’05: Proceedings of the 2005 ACM workshop on Storage security and survivability, ACM, pp. 74–83 (2005).
[8] R. Fagin, J. Nievergelt, N. Pippenger and H. R. Strong: “Extendible hashing—a fast access method for dynamic files”, ACM Trans. Database Syst., **4**, 3, pp. 315–344 (1979).
[9] B. H. Bloom: “Space/time trade-offs in hash coding with allowable errors”, Communications of the ACM, **13**, pp. 422–426 (1970).